Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Vremenska postaja na STM32F4

Vhodno-izhodne naprave

Bodan Temelkovski

Mentor:

viš. pred. dr. Robert Rozman

Univerza *v Ljubljani*
Fakulteta *za računalništvo*
in informatiko

Vsebinsko kazalo

1. Uvod

Poznavanje temperature in relativne vlažnosti v prostoru je ključno za ustvarjanje prijetnega in zdravega bivalnega okolja. V tem poročilu bo predstavljen inovativni projekt, ki se osredotoča na merjenje temperature in vlažnosti s pomočjo senzorja DHT11, mikrokrmilnika STM32F4 in OLED zaslona.

Pravilna regulacija temperature v prostoru neposredno vpliva na dobro počutje in kognitivne sposobnosti. Idealna povprečna temperatura v bivalnih prostorih je med 20 °C in 22 °C, kar zagotavlja udobje in optimalne pogoje za delovanje uma. Zato je natančno merjenje temperature in ustrezno ukrepanje ključnega pomena za ustvarjanje optimalnega bivalnega okolja.

Poleg temperature ima tudi relativna vlažnost pomembno vlogo pri zagotavljanju zdravega okolja. Previsoka ali prenizka vlažnost lahko povzroči nelagodje, težave z dihanjem in vpliva na kakovost zraka, kar lahko negativno vpliva na zdravje. Zato je natančno merjenje in nadzor relativne vlažnosti ključno za zagotavljanje optimalnih pogojev bivanja.

Vezava na github: https://github.com/BodanT/Vremenska-Postaja-STM32F4

2. Seznam potrebnih delov

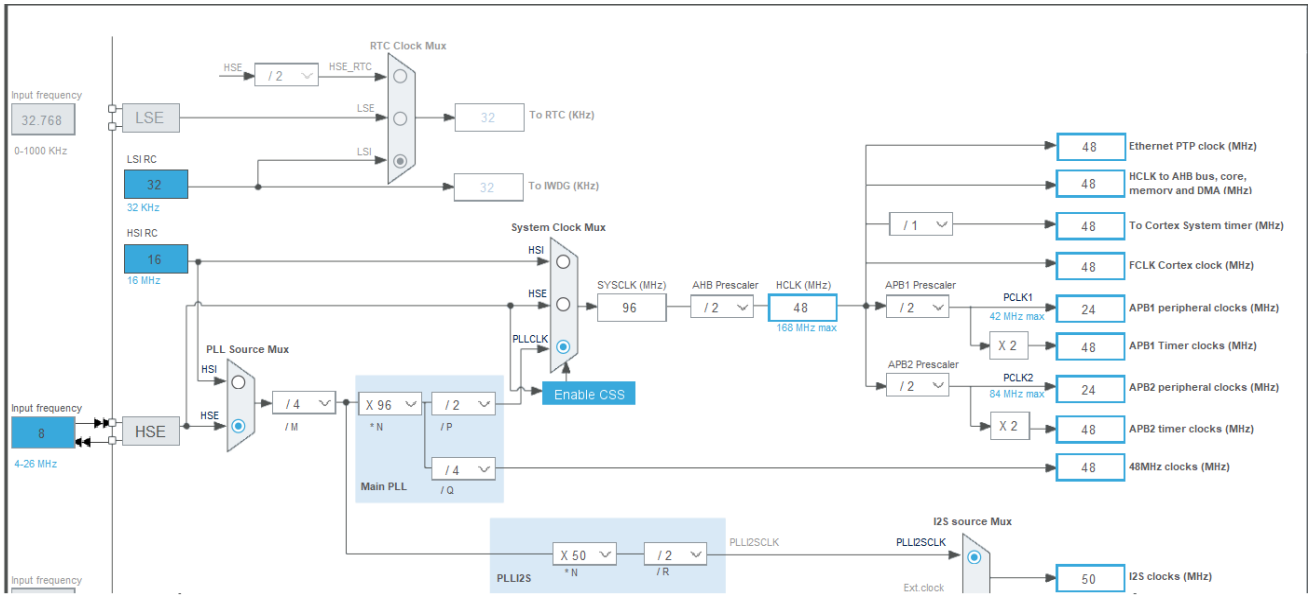
- Mikrokrmilnik - STM32F407 Discovery Kit
- Senzor za merjenje temperature in vlažnosti - DHT11
- Zaslon – SSD1306 OLED
- Breadboard
- Moške in ženske žičke

3. Nastavitve mikrokrmilnika

Pred začetkom programiranja projekta je ključno pravilno nastaviti konfiguracijo mikrokrmilnika. To vključuje določitev uporabljenih pinov, načinov komunikacije, na primer SPI, ter določitev, kateri pini bodo vhodi in kateri izhodi. Poleg tega je potrebno pravilno nastaviti notranji clock in določiti frekvenco delovanja. Za izvedbo teh korakov se priporoča uporaba CubeIDE, ki omogoča grafični vmesnik in enostavno izvedbo potrebnih nastavitev.

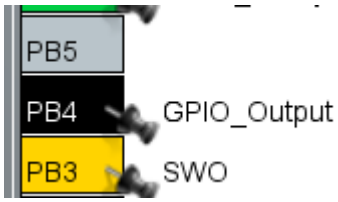
3.1 Clock

Za pravilno delovanje kode, ki uporablja funkcijo delay(), je potrebno pravilno nastaviti notranji clock. V tem projektu je bil izbran TIM6 z naslednjimi nastavitvami: SYSCLK s frekvenco 96 MHz in HCLK s frekvenco 48 MHz. Glede na te nastavitve bodo APB2 timerji delovali s frekvenco 48 MHz, vključno s TIM6.



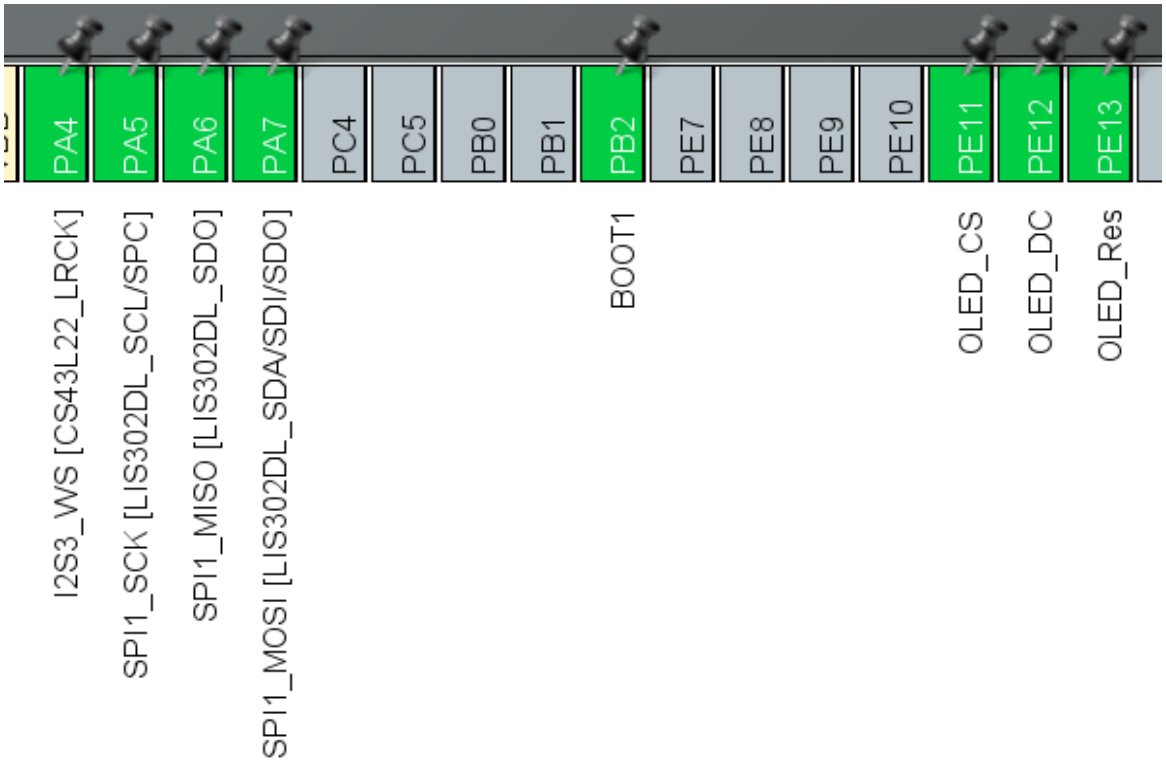
3.2 DHT11

DHT11 senzor sestoji iz treh pinov: Vcc, GND in pin za vhod/izhod. Na mikrokrmilniku je potrebno nastaviti le en pin za povezavo s senzorjem. Za ta namen je bil izbran pin PB4 in je bil nastavljen kot GPIO_OUTPUT. Med delovanjem pa bo ta pin uporabljen tako kot vhodni kot tudi izhodni pin, kar se bo dinamično spreminjalo.



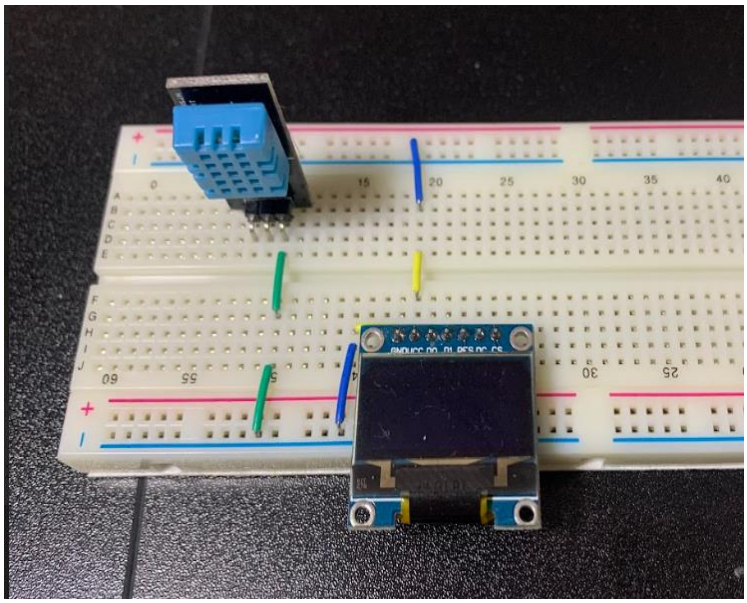
3.3 OLED zaslon

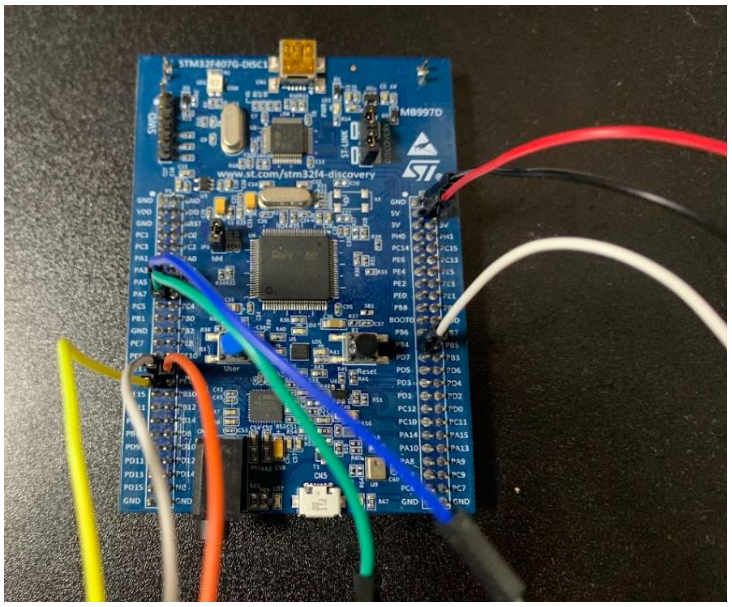
Za vzpostavitev komunikacije med zaslonom in mikrokrmilnikom se v tem projektu uporablja protokol SPI. Zaslon ima skupno sedem pinov: Vcc, GND, D0, D1, RES, DC in CS. Za nastavitev SPI komunikacije na mikrokrmilniku je potrebno uporabiti CubeIDE. V delu povezljivosti (Connectivity) je enostavno omogočiti SPI1 protokol, ki bo samodejno aktiviral pine SPI1_SCK, SPI1_MISO in SPI1_MOSI ter ustrezne pine PA5, PA6 in PA7. Poleg tega pa bodo potrebni še dodatni trije pini. Za ta namen so bili priključene pine PE11, PE12 in PE13, ki so označeni kot OLED_CS, OLED_DS in OLED_Res.

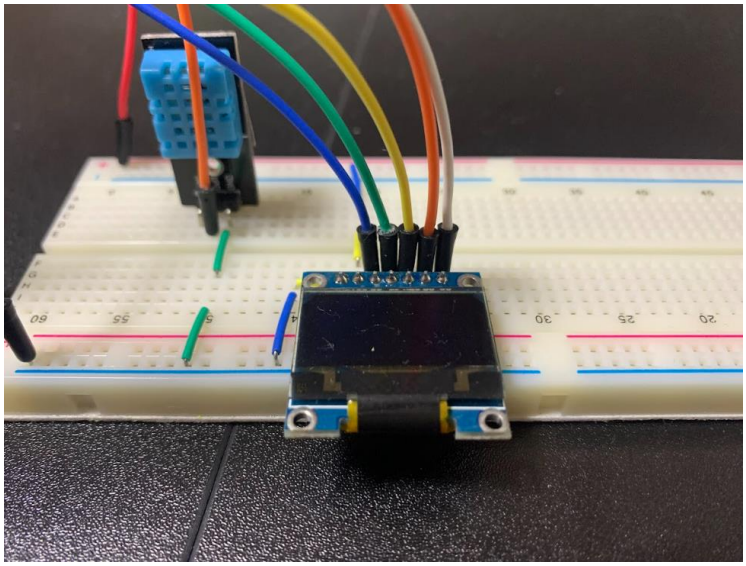


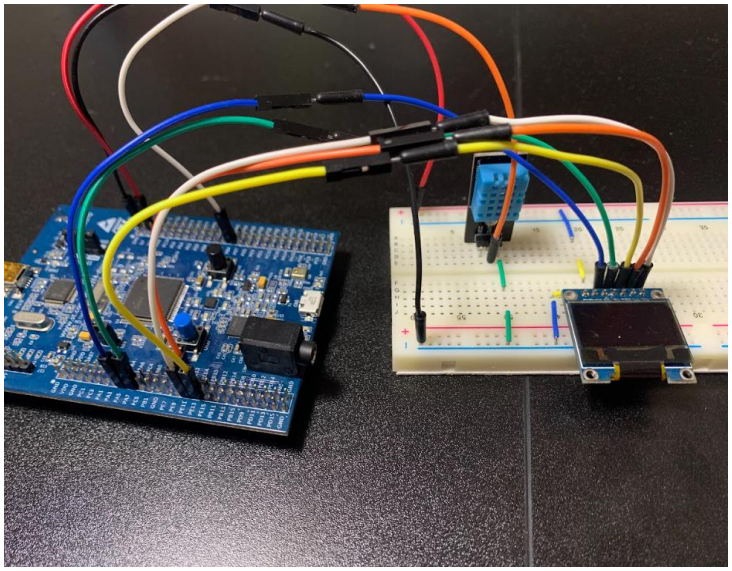
4. Povezave na bredboardu in mikrokrmilniku

Vezava žic na breadboardu in mikrokrmilnika je zelo enostavna. Samo je potrebno slediti zgornjo omejene nastavitve pinov in ustrezno postaviti žice. Naj prej je potrebno nastaviti zaklenjen tokokrog. Vcc je tu 5V in uporabimo rdečo žičko, a GND je črna žička. Za DHT11 vhod/izhod se uporablja oranžno žičko, ki gre v PB4. Za zaslon D0, D1, RES, DC, CS so modra, zelena, rumena, oranžna in na koncu bela žička. Vse te žičke je potrebno ustrezno namestiti na mikrokrmilniku kot je bilo zgoraj omejeno.









5. Delay()

Funkcija delay, ki se uporablja v projektu za DHT11 senzor, omogoča ustrezno zakasnitev. Spreminjanje kode v tej funkciji omogoča nastavitev želene zakasnitve v nanosekundah. S pomočjo funkcije __HAL_TIM_SET_COUNTER(&htim6, 0) se ponastavi števec časa mikrokrmilnika TIM6 na vrednost 0. Nato se izvede zanka while, ki preverja, ali je vrednost števca časa manjša od določene vrednosti time. Funkcija bo zadržala izvajanje programa, dokler se vrednost števca ne doseže želene vrednosti time. S tem se doseže ustrezno zakasnitev, ki omogoča pravilno delovanje DHT11 senzorja v projektu.

```
void delay (uint16_t time)
{
	__HAL_TIM_SET_COUNTER(&htim6, 0);
	while ((__HAL_TIM_GET_COUNTER(&htim6))<time);
}
```

6. Koda DHT11 Senzorja

Za pravilno programiranje DHT11 senzorja je ključno, da programer razume njegovo delovanje in specifikacije. Zato je potrebno temeljito preučiti datasheet, ki je na voljo na naslednji povezavi: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf. V datasheetu so zbrani vsi podatki in informacije o delovanju senzorja, kar je omogočilo pravilno konfiguracijo in uporabo v projektu.

6.1 Pomožne funkcije

Te funkcije se uporabljajo za preklop med vhodom in izhodom

```
void Set_Pin_Output (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
{
	GPIO_InitTypeDef GPIO_InitStruct = {0};
	GPIO_InitStruct.Pin = GPIO_Pin;
	GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
	GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
	HAL_GPIO_Init(GPIOx, &GPIO_InitStruct);
}

void Set_Pin_Input (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
{
	GPIO_InitTypeDef GPIO_InitStruct = {0};
	GPIO_InitStruct.Pin = GPIO_Pin;
	GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
	GPIO_InitStruct.Pull = GPIO_PULLUP;
```

6.2 DHT11_Start

Za uspešno inicializacijo senzorja je ključno izvesti ustrezne korake. Najprej je treba nastaviti pin PB4 kot izhod in ga postaviti v nizko stanje ter počakati 18 ms. Nato se pin preklopi v visoko stanje in znova počaka, tokrat 20 ns. S tem postopkom senzor postane aktiven, nato pa se ga pripravi na sprejemanje podatkov tako, da se pin PB4 nastavi kot vhod.

```
void DHT11_Start (void)
{
	Set_Pin_Output (DHT11_PORT, DHT11_PIN);
	HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 0);
	delay (18000);
    HAL_GPIO_WritePin (DHT11_PORT, DHT11_PIN, 1);
	delay (20);
	Set_Pin_Input(DHT11_PORT, DHT11_PIN);
}
```

6.3 DHT11_Check_Response

Na začetku funkcije se izvede zamik (delay) trajanja 40 mikrosekund. Ta zamik je potreben, saj senzor DHT11 zahteva časovni interval med prehodom iz visoke v nizko stanje pina DHT11_PIN in nazaj.

Nato se preveri stanje pina DHT11_PIN na DHT11_PORT. Če je pin nizke vrednosti, to pomeni, da senzor DHT11 je pripravljen posredovati podatke. V tem primeru se izvede dodaten zamik (delay) trajanja 80 mikrosekund, kar je potrebno za pravilno zajemanje podatkov s senzorja.

Po zamiku se ponovno preveri stanje pina DHT11_PIN. Če je pin visoke vrednosti, senzor je uspešno posredoval podatke. V nasprotnem primeru se vrednost Response nastavi na -1, kar označuje napako pri odzivu senzorja.

```
uint8_t DHT11_Check_Response (void)
{
	uint8_t Response = 0;
	delay (40);
	if (!(HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
	{
		delay (80);
		if ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)))
			Response = 1;
		else
			Response = -1;
	}
	while ((HAL_GPIO_ReadPin (DHT11_PORT, DHT11_PIN)));

	return Response;
}
```

6.4 DHT_Read

Funkcija DHT11_Read je namenjena branju podatkov iz senzorja DHT11. V tej funkciji se izvede zanka, ki se ponovi 8-krat, saj senzor DHT11 posreduje 8-bitne podatke. V vsakem koraku zanke se najprej počaka, da pin DHT11_PIN preide v visoko stanje. Nato se počaka 40 mikrosekund. Naslednje dejanje je branje stanja pina DHT11_PIN. Če je pin v visokem stanju, se ustrezni bit nastavi v spremenljivko data. To se naredi s premikom bita v levo za ustrezno število mest. Po tem se počaka, da pin DHT11_PIN preide v nizko stanje. Celoten postopek se ponovi 8-krat, dokler se ne preberejo vsi podatki. Na koncu se vrne spremenljivka data, ki vsebuje prebrane podatke.

```
uint8_t DHT11_Read (void)
{
    uint8_t data = 0;
    for (uint8_t i = 0; i < 8; i++)
    {
        while (!(HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN)));
        delay(40);
        if (HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN))
        {
            data |= (1 << (7 - i));
        }
        while (HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN));
    }
    return data;
}
```

7. Zaslon

Za upravljanje zaslona je bila uporabljena že obstoječa knjižnica za STM32 mikrokrmilnike, ki je na voljo na naslednji povezavi:

https://github.com/afiskon/stm32-ssd1306.

Knjižnica vsebuje vse potrebne funkcije za učinkovito delovanje programa. Poleg tega knjižnica omogoča uporabo različnih vrst zaslonov, med katerimi je tudi SSD1306 OLED. Enostavno je razpakirati in prenesti vse potrebne datoteke ter izbrati želeni komunikacijski protokol, saj knjižnica podpira tako I2C kot tudi SPI. Za podrobnejše razumevanje delovanja knjižnice je na voljo tudi video posnetek, ki je dostopen na naslednji povezavi:

https://www.youtube.com/watch?v=z1Px6emHIeg.

8. Glavni del programa – main()

Glavni del programa se nahaja znotraj zanke while (1), ki omogoča nenehno izvajanje kode. V tem delu programa se najprej zažene DHT11 senzor s funkcijo DHT11_Start(), nato pa se preveri prisotnost odziva senzorja s funkcijo DHT11_Check_Response(). Če je prisotnost potrjena, se nadaljuje z branjem podatkov senzorja, kot so temperatura in vlažnost, preko funkcije DHT11_Read(). Prebrani podatki se nato pretvorijo v ustrezne vrednosti in shranijo v spremenljivke Temperature in Humidity.

Nato se inicializira SSD1306 zaslon s funkcijo ssd1306_Init(), ki omogoča prikazovanje podatkov. Tekstovne spremenljivke, kot so humidity, temp in naslov, se formatirajo in ustrezno pripravijo za prikaz na zaslonu preko funkcije sprintf().

Sledijo nastavitve položaja prikaza teksta na zaslonu s funkcijo ssd1306_SetCursor(), nato pa se z uporabo funkcije ssd1306_WriteString() prikažejo tekstovne informacije, kot so naslov postaje, temperatura in vlažnost, na zaslonu.

Na koncu se posodobitve na zaslonu prikažejo preko funkcije ssd1306_UpdateScreen(). Če prisotnost odziva senzorja ni potrjena, se na zaslonu prikaže sporočilo "ERROR".

Celoten postopek se nato ponovi z odmikom 20 sekund, kar omogoča periodično branje in prikazovanje podatkov s senzorja na zaslonu.

```
while (1)
{
	DHT11_Start();
	Presence = DHT11_Check_Response();

		if(Presence==1){
			Rh_byte1 = DHT11_Read ();
			Rh_byte2 = DHT11_Read ();
			Temp_byte1 = DHT11_Read ();
			Temp_byte2 = DHT11_Read ();
			SUM = DHT11_Read();

			TEMP = Temp_byte1;
			RH = Rh_byte1;


			Temperature = (float)TEMP + ((float)Temp_byte2 / 10.0);
			Humidity = (float)RH + ((float)Rh_byte2 / 10.0);


			ssd1306_Init();

			char humidity[10];
			sprintf(humidity, "RH:   %.1f", Humidity);

			char temp[15];
			sprintf(temp, "Temp: %.1f", Temperature);

			char naslov[20];
			sprintf(naslov, "Vremenska Postaja");


			ssd1306_SetCursor(0,0);
			ssd1306_WriteString(naslov, Font_7x10, SSD1306_COLOR_WHITE);

			ssd1306_SetCursor(0,18);
			ssd1306_WriteString(temp, Font_11x18, SSD1306_COLOR_WHITE);


			ssd1306_SetCursor(0, 45);
			ssd1306_WriteString(humidity, Font_11x18, SSD1306_COLOR_WHITE);

		    ssd1306_UpdateScreen();

		}else{
		     char problem[20];
		     sprintf(problem, "ERROR");
		     ssd1306_WriteString(problem, Font_11x18, SSD1306_COLOR_WHITE);
		}

		HAL_Delay(20000);
}
```

9. Prikaz delovanje